A COLLABORATIVE TOOL FOR THE INSPECTION OF THREE DIMENSIONAL DESIGN

by

CARRIE A. MORTON
Bachelor of Science in Civil and Environmental Engineering
MIT, 1996

Submitted to the Department of Civil and Environmental Engineering
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF ENGINEERING
IN CIVIL AND ENVIRONMENTAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1997

© 1997 Carrie A. Morton
All rights reserved

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Signature of the Author ______________________________
Department of Civil and Environmental Engineering
May 9, 1997

Certified by ______________________________
Professor John R. Williams
Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by ______________________________
Professor Joseph Sussman
Chairman, Department Committee on Graduate Studies

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUN 24 1997

Eng.

LIBRARIES

A Collaborative Tool for the Inspection of Three Dimensional Design

by
Carrie A. Morton

Submitted to the Department of Civil and Environmental Engineering
on May 9, 1997, in partial fulfillment of the requirements for the degree of
Master of Engineering in Civil and Environmental Engineering

ABSTRACT

The modern era of the global marketplace has been defined by internet communication, long-distance collaboration, and the virtual office space. In an effort to remain competitive, we have to demand the best resources available for everything. In order to meet these needs, professionals need quality tools to connect them to the best in their field, to aid in collaboration and for the visual presentation of their designs and ideas.

The purpose of this text is to define and describe the implementation used to create a collaborative tool for the inspection of three dimensional design using Java and VRML. Explanation will be given for why such a tool is needed and case studies will be presented demonstrating the effectiveness of its use. Ideas for future expansion of the software program are also given.

Thesis Supervisor: Prof. John R. Williams
Professor of Civil and Environmental Engineering

Table of Contents

ABSTRACT **2**

TABLE OF CONTENTS **3**

LIST OF FIGURES **5**

1.0 INTRODUCTION **6**

2.0 TECHNICAL APPROACH **7**

2.1 THEORETICAL BACKGROUND 7

2.2 TECHNICAL BACKGROUND 8

3.0 IMPLEMENTATION **11**

3.1 NETWORKING PACKAGE 12

3.2 JAVA AND VRML 13

3.3 SERVER DESIGN 15

3.3.1 MCserver.java *16*

3.3.2 MCclientThread.java *17*

3.3.3 MCsocketArray.java *18*

3.3.4 MCarray.java *19*

3.4 CLIENT DESIGN 19

3.4.1 MCclient.java *20*

3.4.2 MCreceive.java *24*

3.4.3 MCsend.java *25*

3.5 USER INTERFACE 26

3.5.1 Chat Text Area *26*

3.5.2 User List Field 27

3.5.3 Input and Action Area 27

4.0 CASE STUDIES 29

4.1 COLLABORATIVE ARCHITECTURAL STUDY 29

4.2 FIRE SAFETY TRAINING 33

5.0 FUTURE DEVELOPMENT 35

5.1 IMPROVEMENTS 35

5.2 VARIATIONS ON DESIGN AND USE 36

6.0 CONCLUSION 38

REFERENCES 39

APPENDIX A: SERVER CODE 40

APPENDIX B: JAVA CLIENT CODE 49

APPENDIX C: VRML CLIENT CODE 59

APPENDIX D: SAMPLE FILES FROM CASE STUDIES 60

List of Figures

FIGURE 2.1A: NETWORKED COLLABORATION 8

FIGURE 3.0A: RELATION BETWEEN MCSERVER, MCRECEIVE, MCSEND, AND MCCLIENT 11

FIGURE 3.2A: EAI VS. SCRIPT NODE 14

FIGURE 3.3A: OBJECT MODEL FOR SERVER IMPLEMENTATION 16

FIGURE 3.4A: GENERALIZED OBJECT MODEL FOR CLIENT IMPLEMENTATION 20

FIGURE 3.5A: NUMBERED USER INTERFACE 26

FIGURE 3.5B: APPLET USER INTERFACE 28

FIGURE 3.5C: VRML USER INTERFACE 28

FIGURE 4.1A: CONVERSION FROM AUTOCAD TO VRML 30

FIGURE 4.1B: SCREEN SHOT OF BUILDING DESIGN USER INTERFACE 32

FIGURE 4.2A: SCREEN SHOT OF FIRE SAFETY TRAINING USER INTERFACE 34

1.0 Introduction

In today's environment of frequent computer use, global offices, and communication via email, there has arisen a desire for methods of collaboration without the need for being in the same office at the same time. When co-workers in offices thousands of miles away can be as close as the nearest computer terminal, time-consuming travel and related expenses will become a thing of the past. In order for this vision to become reality, the proper tools to facilitate these needs must be developed.

Consider the case in which an architect and engineer (one in California, the other in New York) are working together on the design of new building. They wish to collaborate on the drawings, but being on different coast makes it difficult to get together and review the plans. They need a method for viewing the same file at the same time while a channel of communication remains open for them to discuss their ideas and thoughts about this proposed three dimensional object.

Through the use of computers and internet technology, we now have the ability to communicate over vast physical space with virtually no delay and to produce three dimensional rendered images of our designs and ideas. What is needed to produce a collaborative tool that would solve the problems of the architect and the engineer mentioned above is the ability to simultaneously present 3D graphical images, demonstrate to others our viewpoint of these images, and to communicate through a text based interface. The purpose of this paper is to describe in detail the architecture, implementation, and application of just such a tool.

2.0 Technical Approach

This section will explain the motivation behind the technical approach that was taken in creating a collaborative tool for the inspection of three dimensional design. A brief introduction to the methods of implementation, mainly the selection of software tools, will be given.

2.1 Theoretical Background

In the current world of global markets, you need to pool the best resources in the field just to stay competitive. Unfortunately, sometimes these resources are not always in your backyard. Today, nearly every country in the world and nearly every business in existence is connected to the internet in one form or another. With this groundwork, we have the potential for finding the best people in the world for consulting and collaborating on nearly any project we take on.

In the example of the architect and the engineer, in order to collaborate successfully on a design, they need a three dimensional graphics tool and a effective method of long-distance communication. By linking these two together, a powerful tool for networked collaborative inspection can be created. The software tools VRML and Java may be used in execution over the internet. When combined, they meet these qualifications for generating an effective three dimensional collaboration devise.

Figure 2.1a portrays the linking of professionals via computers and the internet.

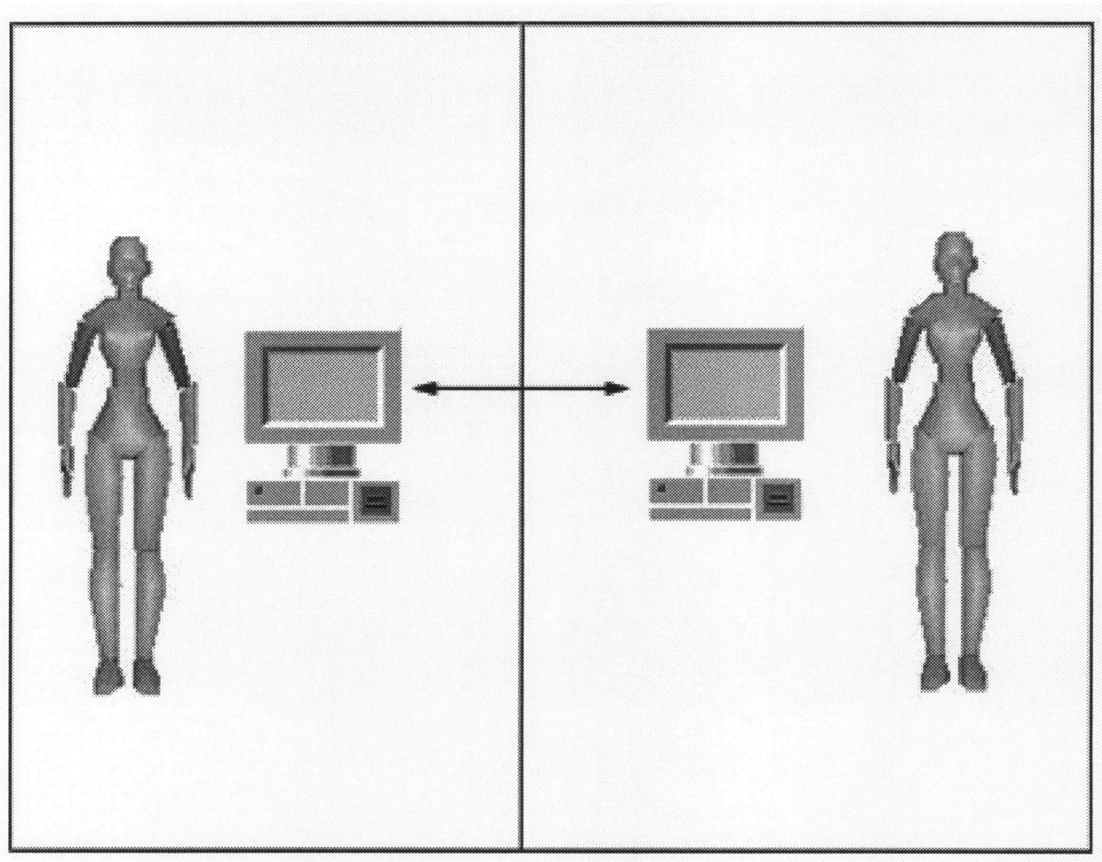

Figure 2.1a: Networked Collaboration With the use of computers and the internet, professionals are able to communicate and collaborate over long distances.

2.2 Technical Background

Virtual Reality Modeling Language (**VRML**) is a scene description language that can be used to model three dimensional worlds. It has been developed and maintained by Silicon Graphics, Inc. for use in modeling and interacting with three dimensional spatial environments. Through the use of a VRML interpreter[1], users can move through and interact with computer generated "virtual worlds" similar to the ways in which we interact with our real environment. This is useful for collaborative inspection because standard computerized drawings (such as

[1] Interpreters are programs that convert VRML text files into three dimensional graphical output. They may be linked to a web browser as a plug-in utility. Examples of such VRML interpreters include Cosmo Player (Silicon Graphics, Inc.), Liquid Reality (Dimension X), and Live 3D (Netscape Inc.).

AutoCAD[2] files) may be converted to VRML for more "life-like" presentation to clients, colleagues, or pupils.[3]

Java is an object oriented programming language developed by Sun Microsystems, Inc. Once compiled, Java programs are also platform independent, which is very useful when designing programs to be downloaded over the internet, where the programmer can not always be certain of what platform the end user will have. Java programs can be either **applications** or **applets**. To run an application on your computer, you need to have the executable program (with a ".class" extension) and then run it with a Java Virtual Machine[4]. Applets are intended to be included in HTML[5] documents and executed via Java enabled web browsers (such as Netscape Navigator or Microsoft Internet Explorer). Applet programs reside on the computer that runs the web server you are requesting information from and are downloaded to a temporary location on your computer for the duration of execution.

Both applications and applets are useful for collaborative inspection. Network servers, which will be run only on the "host" machine, can be applications controlled at the command line. Applets can be run on the client side through web browsers along side VRML interpreters for simultaneous three dimensional viewing and "real-time" network communication. In addition to this, Java programs can interact with VRML files through the External Authoring Interface to pass valuable information from one to the other.[6]

---

[2] AutoCAD is a product of Autodesk, Inc.
[3] For more information on VRML, please see the Silicon Graphics, Inc. web pages at "http://vrml.sgi.com/".
[4] Examples of Java Virtual Machines are java.exe from Sun Microsystems, Inc. and jview.exe from Microsoft Corporation. They are run from the command prompt such as "C:\java MyProgram", where MyProgram.class is an executable java program.
[5] HTML means Hypertext Mark-up Language and is a file format interpreted by web browsers.
[6] For more information about the Java program language, see the Sun Microsystems, Inc. web pages at "http://www.javasoft.com/".

The External Authoring Interface is a method developed by Silicon Graphics, Inc. (SGI) for external programs to access and send information to or from a VRML file. A series of Java classes and methods have been created by SGI for use in linking Java to VRML in this way. These classes may be downloaded from the SGI website when downloading and installing Cosmo Player beta version 3a (a VRML interpreter). For more information on this "Java for VRML" package, see the Proposal for a VRML 2.0 Informative Annex: External Authoring Interface Reference documentation at "http://vrml.sgi.com/moving-worlds/spec/External Interface.html" [Marrin].

3.0 Implementation

This chapter will describe in detail the implementation used for the collaborative VRML inspection tool. Section 3.1 will discuss the integration of the networking package which consists of four main Java class files. The first of these files is **MCserver**, which opens the **ServerSocket** and waits for incoming **Sockets**[7] of information. **MCsend** and **MCreceive** run on the client side and handle the sending and receiving of socket information respectively. The fourth file, **MCclient**, defines the graphical user interface and interacts with both MCsend and MCeceive.

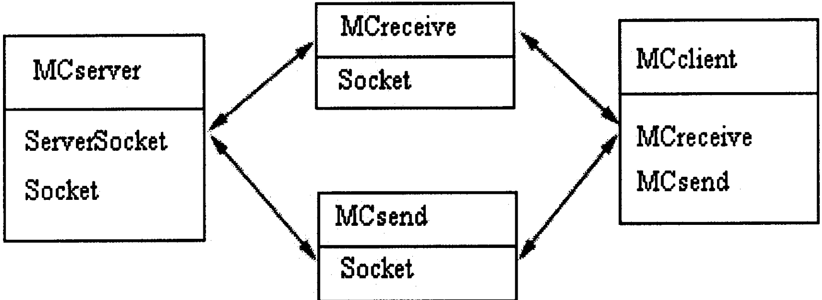

Figure 3.0a: Relation between MCserver, MCreceive, MCsend, and MCclient MCserver contains a ServerSocket and a Socket. MCreceive and MCsend also contain an object of type Socket. MCclient contains an MCreceive object and an MCsend object. Arrows indicate information exchange.

Section 3.2 will discuss the interaction of VRML and Java for the purpose of collaborative inspection. Sections 3.3 and 3.4 will detail the programming code used for the

[7] See Section 3.1 for explanation of ServerSockets and Sockets.

implementation of the server-side program and client-side program respectively. The graphical user interface will be discussed and displayed in section 3.5.

3.1 Networking Package

To implement a live network connection, both server-side and client-side programs needed to be created to handle input and output data streams. The server-side module is run on the host machine and is designed to connect to a specified port on that machine to retrieve all data sent to that port from any location on the internet and to output information through that same port to all clients which may be connected. On the client-side, modules are designed to connect to a specified host machine and port number to communicate information to and from the running server program.

"Socket" is a Java class in the **java.net.*** package from Sun Microsystems, Inc. A Socket is an object containing methods to handle input and output streams through a particular host machine and port number specified as arguments to the Socket constructor function. Sockets are found in both the server and client programs. "ServerSocket" is a class also found in the java.net.* package. ServerSockets are used in the server program only and take as arguments to their constructors the port number to which the server will be connected. Once a ServerSocket has been created, the **accept()** function is used to tell the program to wait indefinitely until a client connects to that port and then create a Socket for all communication between the server and that client (one Socket for each client). All output streams from the client become input streams to the server, as all output from the server becomes input to the client. [Cornell, pg. 555-557.]

3.2 Java and VRML

As mentioned in section 2.2, the object-oriented programming language, Java, and the scene-description language, VRML, can be very useful to the collaborative inspection of three dimensional worlds. By linking these two tools, a better method of interaction with a 3D computer rendered environment may be devised. By taking advantage of Java's object-oriented design and robust networking capabilities, along with VRML's ability to traverse three dimensional spatial environments, a multi-user method for inspecting 3D objects and worlds can be conceived.

Two basic methods for combining VRML with Java are the VRML Script node and the External Authoring Interface (EAI). Nodes are the essential building blocks of all VRML files and Script nodes are special blocks which define either executable programs or JavaScript functions that will be linked to the virtual world. EAI is a protocol and set of executable functions that define allowable interaction between external programs and VRML worlds. [Scott]

In the case of the Script node, the specified program or JavaScript will respond to values sent to the **eventIn** field(s) of the script node. Then, depending on the architecture of the program or script, values will be sent to the Script node's **eventOut** field(s) or fields of other nodes. A Java executable program intended to be used in this manner must "**extend Script**"[8] and therefore adhere to all the properties and behaviors of a proper Script class. In the case of the collaborative inspection tool, we wish to use both Applet and Thread classes (the reasons for this will be explained in section 3.4). To do this with a Script node, information from the server would need to be passed through a Script class to the VRML file and vice versa. This would

---

[8] In Java, to "extend" another class means to inherit from that class.

become cumbersome and slow the process down greatly. To this extent, EAI offers many advantages. (See Figure 3.2a)

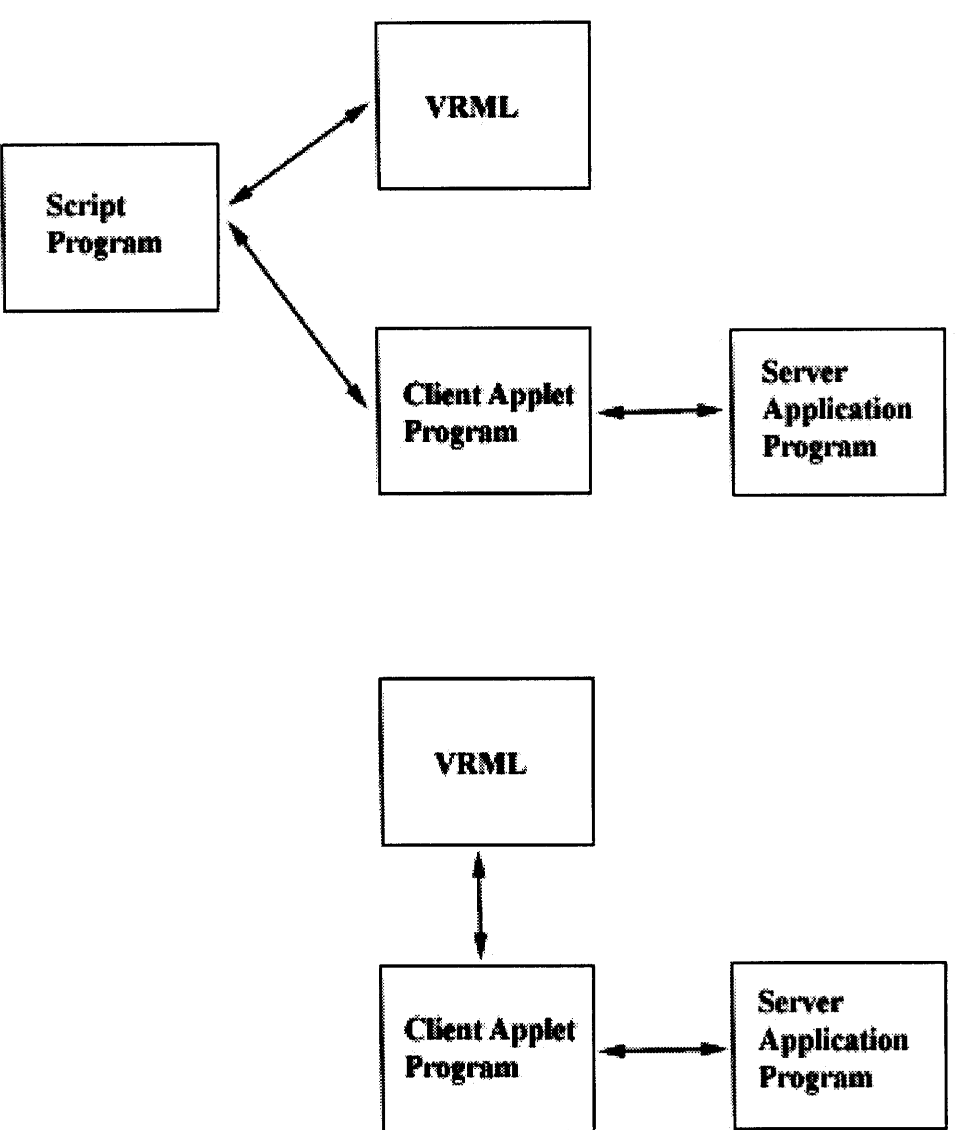

Figure 3.2a: EAI vs. Script Node The Script Node interface (top) is indirect, while the EAI (bottom) links VRML directly to the communication applet program.

With the External Authoring Interface, you can create an HTML document that contains an Applet and one or more embedded VRML files. Through functions contained in the special VRML Java class packages, the Java program retrieves the name and location of these embedded VRML files. With this information, the Java Applet can communicate directly with the VRML files, creating instances of nodes and fields contained in these files to retrieve and manipulate field values as the program designates. This direct connection between Applet and virtual world allows for both a smoother and faster interaction.

Please see Appendix C for a description of the nodes used in the "Java to VRML" interface of this program.

3.3 Server Design

The Java server program runs on the host machine and is responsible for opening a connection to the designated port number and routing all data input Sockets out to all running client applets connected to that specified port and machine. The original server program used for this inspection tool was written by Manuel Perez (M.Eng. EECS MIT '97) on September 5, 1996. No alterations to the server program were required.

The name of the main server program is MCserver.class (for MiniChat server). It imports two other Java programs, MCclientThread.class and MCsocketArray.class, as well as using the standard Java packages **java.net**, **java.io**, and **java.lang.Thread** from the JDK 1.0 from Sun Microsystems, Inc. One additional class is used in the server design. This class is MCarray.class which maintains an array of any object type. This class is used by MCsocketArray.class to sustain an array of client sockets open to the server. See figure 3.3a for the object model of the server classes.

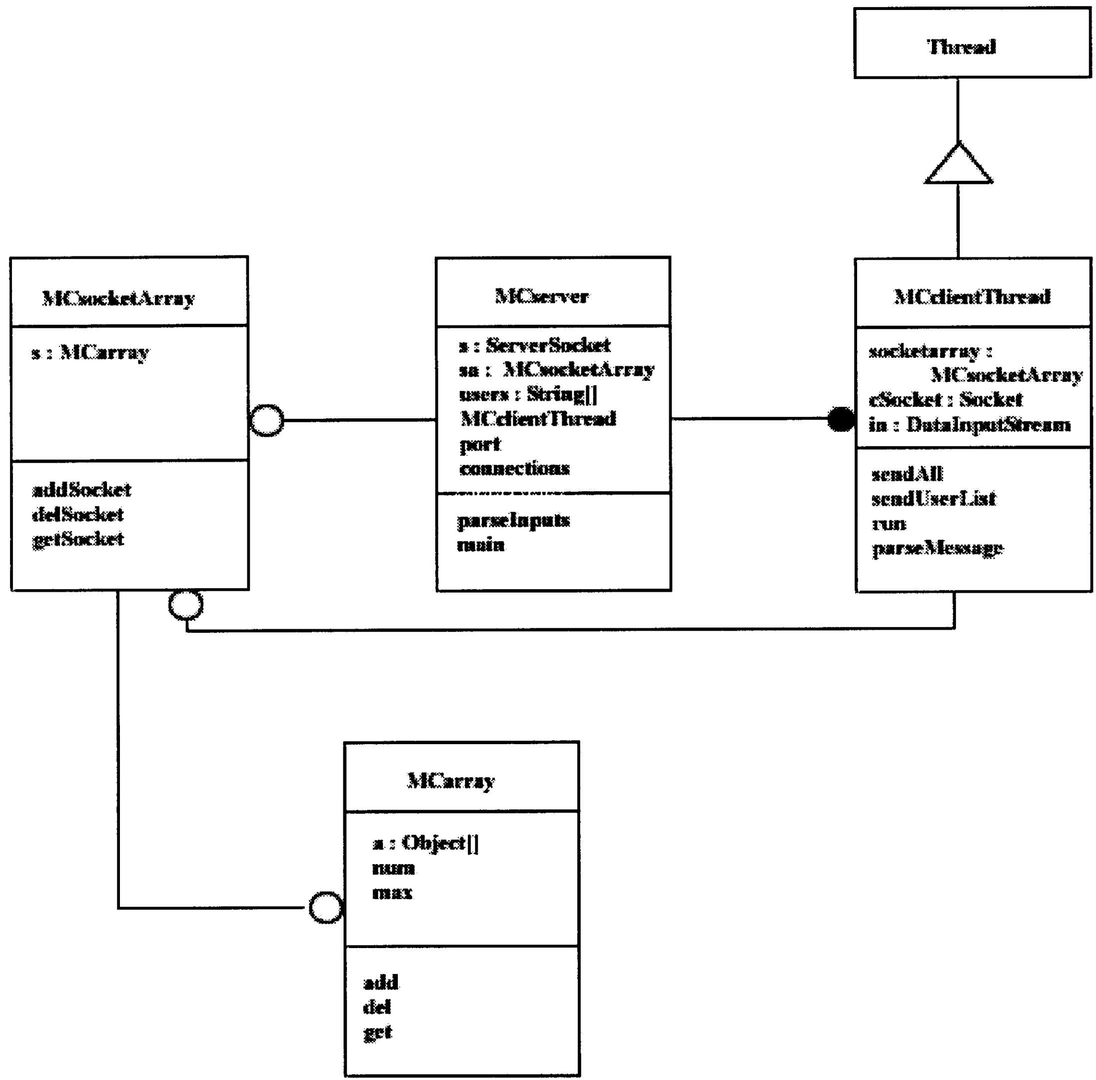

Figure 3.3a: Object Model for Server Implementation This diagram shows the relationship between the four files of the server program.

3.3.1 MCserver.java

MCserver is a Java application which inherits from the generic Java base class **Object**. Arguments passed to the constructor function are the port number the server is expected to communicate through and the maximum number of connections the server should allow at one

time. These inputs are parsed through the **parseInputs(..)** function. If either of these arguments is invalid, parseInputs(..) returns a boolean value of false and the constructor is finished. If the arguments are valid, a try/catch method is used to attempt to open a ServerSocket connection to the correct port. If this attempt is unsuccessful, the boolean variable **started** is set to false.

The programs **main(..)** function is implemented once a user runs the MCserver from the command line. This is where the constructor is called and the user input of port number and maximum allowable connections are passed. If the user does not specify these values, default values are. Throughout the duration of main(..), the server "listens" to the designated port and starts a new MCclientThread every time a new Socket is created.

3.3.2 MCclientThread.java

Class MCclientThread **extends Thread**. This means that it inherits properties from the Java base class "Thread". A thread is considered to be a computational unit or task. Each thread is running its own separate processes and requires memory from the CPU. Once a thread is "started" it runs these processes continuously until suspended, stopped, or told to "sleep". If managed properly, threads can allow to run many continuous processes seemingly at the same time without depleting your computer's memory resources. [Cornell, pg. 501-502] This is useful to the collaborative tool because both the client and the server must listen to the port continuously for incoming and outgoing data streams.

As mentioned earlier, every time a new Socket is created, an MCclientThread is constructed and started. The MCclientThread constructor takes as arguments a Socket, an MCSocketArray, and a String containing all of the current usernames. A try/catch method is used to add the Socket to the Socket array, and to add the default username, "guest", to the end of

the string of usernames. Each time a new client socket is added, the client will be called "guest" until they reset their name through the user interface. Once the user presses the "Set Name" button, a string is passed to MCclientThread with the keyword SETNAME and the string that defines their new name. This message is caught by the MCclientThread class when it encounters the special keyword. The function **parseMessage(..)** then resets the **name** variable of that MCclientThread and updates the string of usernames.

If the keyword SETNAME is not encountered, the **sendAll(..)** function is called which embeds the clients username in the beginning of the passed string. This message is then sent on to the server.

When a thread is started, the **run()** function of the thread is implemented. This function "listens" to the socket connection for incoming streams. If the message is not null, it is parsed and either a new username is set or the sendAll(..) function is called. If MCclientThread does receive a null message, then that client's session has ended and the Socket is closed.

3.3.3 MCsocketArray.java

The MCsocketArray class maintains the array of socket connections between the server and various clients. An object of this type is created when the server's constructor function is implemented. The MCsocketArray is passed to MCclientThread's constructor function in the server's main function each time a new Socket is created.

MCsocketArray's constructor class takes as an argument the maximum number of allowable connections to the server. In this constructor method, a new MCarray object is created with the maximum number of connections being passed as an argument (See section 3.3.4 to see what is done with this number).

Functions **addSocket**(..) and **delSocket(..)** add or remove new or closed Sockets to the MCarray object. The function **getSocket(..)** returns a Socket of interest from the MCarray. This function is implemented from the MCclientThread class.

3.3.4 MCarray.java

The constructor for the MCarray class takes as an argument the maximum number of allowable connections to the server. It then creates an array of type **Object** with that designated number of members. When the MCarray object is first created, there are no actual Sockets contained in the array, so the variable **num** is set to zero.

Functions **add(..)** and **del(..)** add and delete Sockets (treated as base class type **Object**) to the array. If an attempt is made to add a new Socket to the array when there already exists the maximum number of connections, an Exception is thrown and the Object is not added. In both functions, the variable, num, is incremented or decremented accordingly to keep track of the number of current connections. The function get(..) finds the Object of interest (by index number) and returns it to the class from which the function was called.

3.4 Client Design

This section will describe the client side programs, MCclient, MCsend, and MCreceive. MCclient defines the graphical user interface, establishes which host and port to connect to, and controls all Java/VRML interaction. MCsend takes MCclient's output and passes it to the server. MCreceive takes output from the ServerSocket and either reacts to commands embedded in that output, or passes the string along to the MCclient class. See figure 3.4a for a generalized object model of the interaction of client classes.

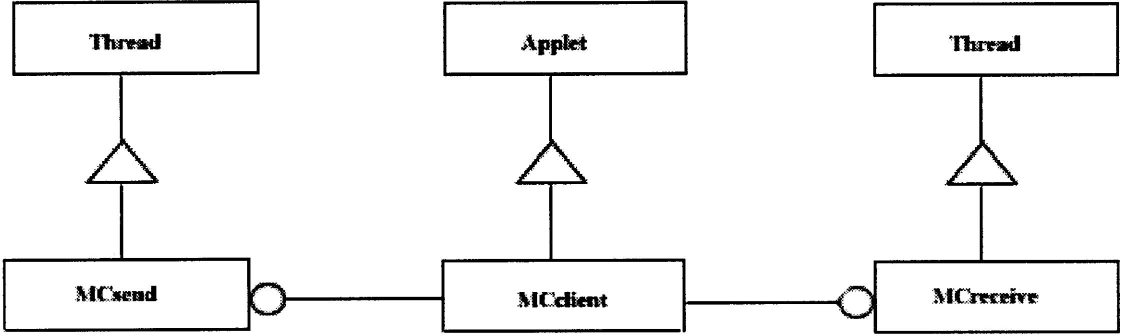

Figure 3.4a: Generalized Object Model for Client Implementation This is a generalized representation of the interaction of the three client classes.

3.4.1 MCclient.java

The MCclient class **extends Applet** and **implements EventOutObserver**. The term "extends Applet" means that MCclient is **derived** from the existing class **Applet**. It will inherent behaviors and functions from this base class. Most of the functions that define the graphical user interface come from the Applet class.

To say that MCclient "implements EventOutObserver" means that MCclient is expected to use functions declared in class EventOutObserver. In this particular case, the function used by MCclient is

void callback(EventOut value, double timeStamp, Object userData)

which, while declared in vrml.external.field.EventOutObserver.java, is defined in MCclient.java. In the initialization of MCclient, the **advise**(..) function from the EventOut class is called to notify EventOutObserver which EventOut field to monitor. Every time a value is sent in VRML

to that particular EventOut field, the callback function is called. This is where most of the VRML/Java interaction occurs.

When an applet is loaded into a web browser, that applet's init() function is called automatically. In the first section of MCclient's init() function, information is retrieved from the HTML file from which it was loaded (see Appendix B). This information is used to create instances of the web browser window, the HTML document, and the VRML file which is embedded in the document. This is necessary so that the Java program knows which VRML world to interact with.

The next process in the init() function is a try/catch method which attempts to create instances of particular nodes in the VRML file. These nodes, **ZERO** and **MOTIONTRACKER**, are used to retrieve changes in one user's position in the environment (MOTIONTRACKER) and output these values to other users' viewpoint (ZERO). **position_changed** is an instance of MOTIONTRACKER's EventOut position_changed. When this member's **advise(..)** function is called, the EventOut class notifies the EventOutObserver class that any changes to this variable will cause the callback function to be implemented[9]. If the try method is not successful, catch will output the error to the Java Console. There is a while loop surrounding the try/catch method to ensure that it is repeated until the VRML file is fully loaded into memory and all nodes of interest can be found. In the event that one or both of these events can not complete successfully, the program will continue in an infinite loop until the web browser window is terminated. Therefore, it is important to check that the designated file can be

[9] Refer to above discussion of classes EventOut and EventOutObserver and the callback() function.

found in the specified directory and that both ZERO and MOTIONTRACKER[10] nodes are present.

The next process that occurs is MCclient's init() function is that the user interface is created. Standard applet objects, such as BorderLayout and Panels are used. For more information about the graphical user interface, see section 3.5.

The final initialization process is to establish the network connection. **host** and **port** variables are taken from parameters specified in the HTML document from which the applet was loaded. It is important to note here that due to security measures of most Java-enabled web browsers, an applet may only connect to the host machine from which it was downloaded. With this information, instances of the MCreceive and MCsend class are created. Due to the fact that these classes both extend Thread (see sections 3.4.2 and 3.4.3 and refer to discussion in section 3.3.2), the threads are then started.

The next important function in the MCclient class is **callback(...)**. As mentioned earlier, this function is declared in the EventOutObserver class and defined here in MCclient. The function is called every time a change in **position_changed**'s value is detected. This function retrieves the new position and orientation values from the MOTIONTRACKER node of the user. If the user happens to be in control of the viewer[11], that is to say the one that determines the viewpoint of all other users, a special message is sent to the server containing the keyword "**MOVEMENT**" and the position and orientation values are retrieved. Because messages passed through the server are of type String, the position and orientation vectors are broken down into individual components, each separated by a space. They will be converted back into vectors when the string is received by other clients.

---

[10] ZERO must be a node of type Viewpoint and MOTIONTRACKER must be a ProximitySensor.

The function **MoveWorld(..)** is called from class MCreceive when it encounters the keyword MOVEMENT. The argument passed to this function is the remainder of the string sent from the server that follows the keyword. As mentioned above, the remainder of this string is the individual components of the position and orientation variables received from the ProximitySensor, MOTIONTRACKER. The function converts this string to two vectors of floats by subdividing the string into sections that are separated by spaces and then converting that value to type float. The first three values become the x, y, and z components of the position vector, while the remaining four are the x, y, z, and θ of the orientation vector, respectively. The last job of the MoveWorld function is to send these vectors to the input of the viewpoint node, ZERO, for all clients save the one in control of the viewer (that user's viewpoint has already been changed manually).

In order to synchronize the views of all clients, it is important that only one user be in control of the viewer at a time. For this reason, the variable, **Control**, was established. Initially, all client's Control variables are set to zero (meaning *not* in control). When a user hits the "**Take Control**" button on the applet interface, their Control variable is set to one (*in* control) and a message is passed through the server with the keyword TAKINGCONTROL. Once MCreceive finds this keyword in a string from the server, the function **ChangeControl(..)** is called for all clients save the one that sent the message. The argument passed to this function is the entire string received from the server. Whenever the server sends out a message, it embeds the username[12] of the client who sent that message, preceded by a "<" symbol and followed by a ">" symbol. This occurs at the very beginning of the string. Using the substring(..) method of the

[11] Control of the viewer will be discussed later in this section.
[12] By "username" it is meant the name that the user specifies in the Set Name text field of the applet. Due to Java applet security, the applet is unable to retrieve the actual username of the person running the program.

String class, the username is abstracted and used to send a string to the client's chat text area notifying them of which user has taken control of the viewer. Before this message is sent, however, the client's Control variable is set to zero.

3.4.2 MCreceive.java

Class MCreceive **extends Thread**. This means that it will inherit behaviors and functions from the Java base Thread class. The MCreceive constructor takes as arguments the host name and port number retrieved in MCclient, along with an instance of the MCclient class that created the MCreceive object. A try/catch method is used to create an open Socket to the specified host and port. If no connection is made, the boolean variable "**loop**" is set to false and the program will not attempt to read input from the server.

Once the MCreceive object is started (with the start() function called from the MCclient class), the **run**() function is implemented. Again a try/catch method is used to avoid encountering unrecoverable errors when non-valid memory is accessed. A DataInputStream is attempted to be opened between the client and the server. If the connection was successfully made in the constructor (loop is true), a continuos while loop is run searching for incoming data from the server. This input is then passed to a parsing function (described below) to determine if any special instructions are required. If no keywords are detected, the message is then passed on to the MCclient class.

The three keywords used in the collaborative inspection tool are "**USERLIST**", "**MOVEMENT**", AND "**TAKINGCONTROL**". These keywords were chosen to correspond to the special events of updating the user list, detection of movement in the controller's viewer, and one client user taking control of the viewer. It is assumed that these words will not appear as

the are (capital letters, and in some cases, two words together with no spaces) in the average chat conversation between users. In the event that a user does use one of the words, as they appear, in text sent to the server, the string will be treated as a special instruction to the client and be treated accordingly[13].

If the keyword, USERLIST, is detected, the remainder of the string that follows this word will be sent to the function **setUsers(..)** of the MCclient class[14]. If MOVEMENT is detected, the function MoveWorld(..) is called with the substring following the keyword passed as an argument. When TAKINGCONTROL is found in a string, the entire string is passed to the ChangeControl(..) function.

3.4.3 MCsend.java

The class MCsend also **extends Thread**. Its constructor takes as arguments the socket created in the MCreceive class and an instance of the MCclient class that created the MCsend object. Once the MCsend thread has been started, the run() function is implemented.

In the try method of the run() function, a DataOutputStream to the server is opened. If MCclient has no message to be sent, the thread *sleeps*[15] for 5 milliseconds and then checks again for any outgoing message. If such a message is detected, MCsend tells MCclient to reset its outgoing message to null (so that the same message will not be sent twice) and then sends the message it did receive to the server. Once this is done, the MCclient class is told to repaint its window so that all changes will be displayed to the user.

---

[13] In some cases, this could cause an invalid access of memory and the client's program will crash.
[14] See Appendix B for the setUsers(..) function definition.
[15] When a thread "sleeps" it remains idle for the duration of time specified, allowing other processes the opportunity to be executed.

3.5 User Interface

The graphical user interface of the collaborative inspection tool is comprised of a Java applet window embedded in an HTML page beneath a VRML file (run by an appropriate VRML viewer program)[16]. The applet window is divided into three main parts: the chat text area, the user list field, and the input and action area. See figure 3.5a below.

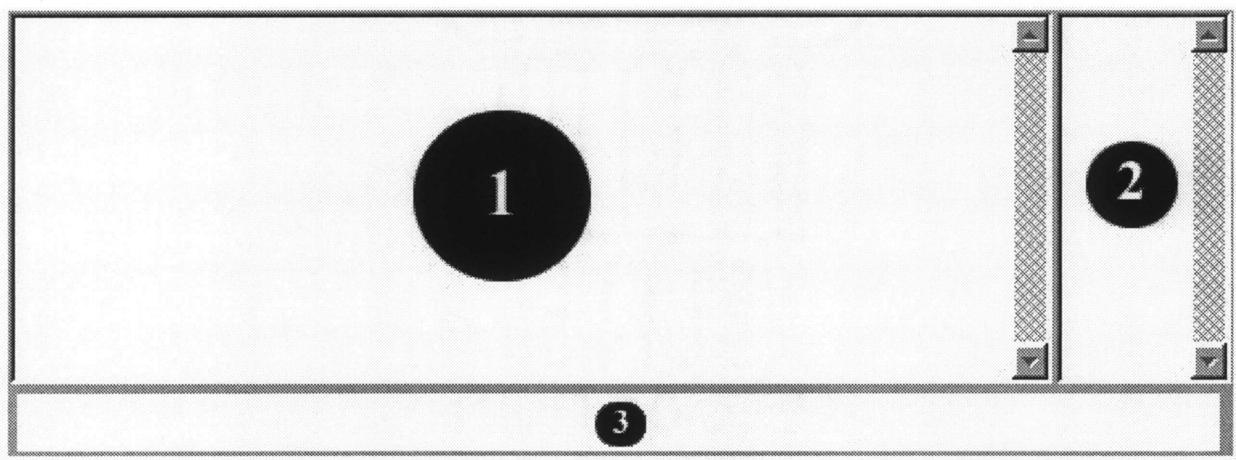

Figure 3.5a: Numbered User Interface 1. Chat Text Area 2. User List Field 3. Input and Action Area.

3.5.1 Chat Text Area

The chat text area, shown as number 1 in Figure 3.5a, is a non-editable text area[17]. This means that the user is not allowed to write to this field, but text can be set through the Java applet. In this area, all messages received by the client that do not contain special instruction keywords will be displayed in this area. Because a client does not receive its own messages that it sends out, the Java applet sends these directly to the chat text area. Whenever control of the

---

[16] See figure 4.1b for a screen capture of what this will look like in a web browser.
[17] **TextArea** is a member of the Applet Java base class.

viewer changes hands, a message is displayed in this area notifying the users of who has taken control.

3.5.2 User List Field

Like the chat text area, the user list field (number 2 in Figure 3.5a) is also a non-editable text area. The values of this field are set by the MCreceive class when the USERLIST keyword is detected. The MCclientThread sends the string of usernames continuously (see code in Appendix A for MCclientThread run() function), so the values in this text area are always current.

3.5.3 Input and Action Area

This area, at the bottom of the applet (number 3 in Figure 3.5a), is where all non-VRML user input to the applet occurs. See figure 3.5b for a screen capture of the applet. The first text field (on the left) is used to enter text that will be sent to the chat text area. This will be sent when the user either hits the return key on the keyboard or the **Send** button on the applet window. The next text field is used to set the users name. The value in this field will be sent to the server once the user hits the **Set Name** button on the window. The values in both of these text fields will be cleared once the command to send the information has been received to assure the user that the strings have been sent to the server.

The last button on the Input and Action Area is the **Take Control** button. This is used when a user wishes to show his/her view of the virtual world to all others connected to the same server. This button also sends a message notifying other users of who has taken control.

Figure 3.5b shows the actual Java applet user interface as it appears on the computer screen. An example of the VRML interface is displayed in figure 3.5c. This particular example uses Cosmo Player beta 3a, from Silicon Graphics, Inc., run on a Window NT workstation. All VRML movement input occurs through the control panel of the chosen interpreter.

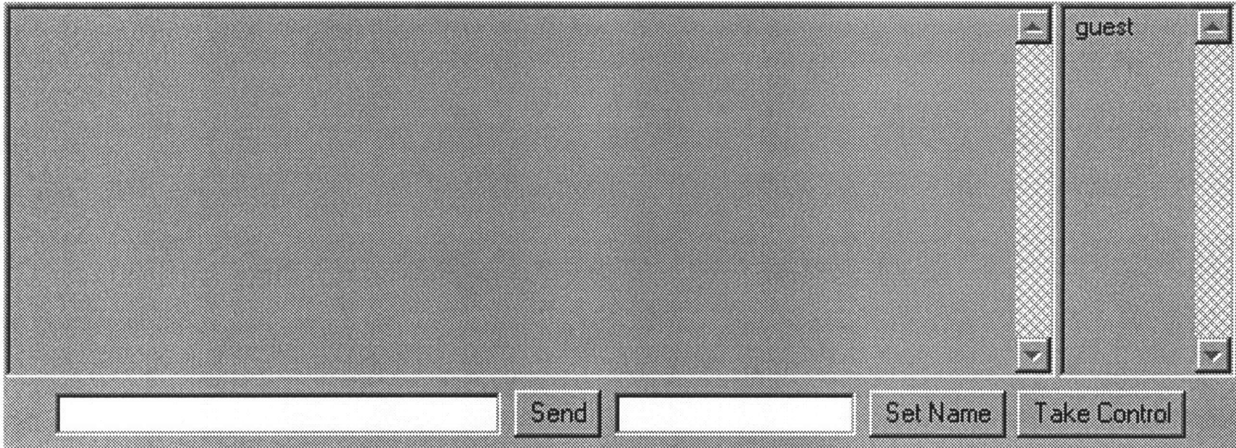

Figure 3.5b: Applet User Interface This is the actual user interface of the Java applet as it appears in the user's browser window.

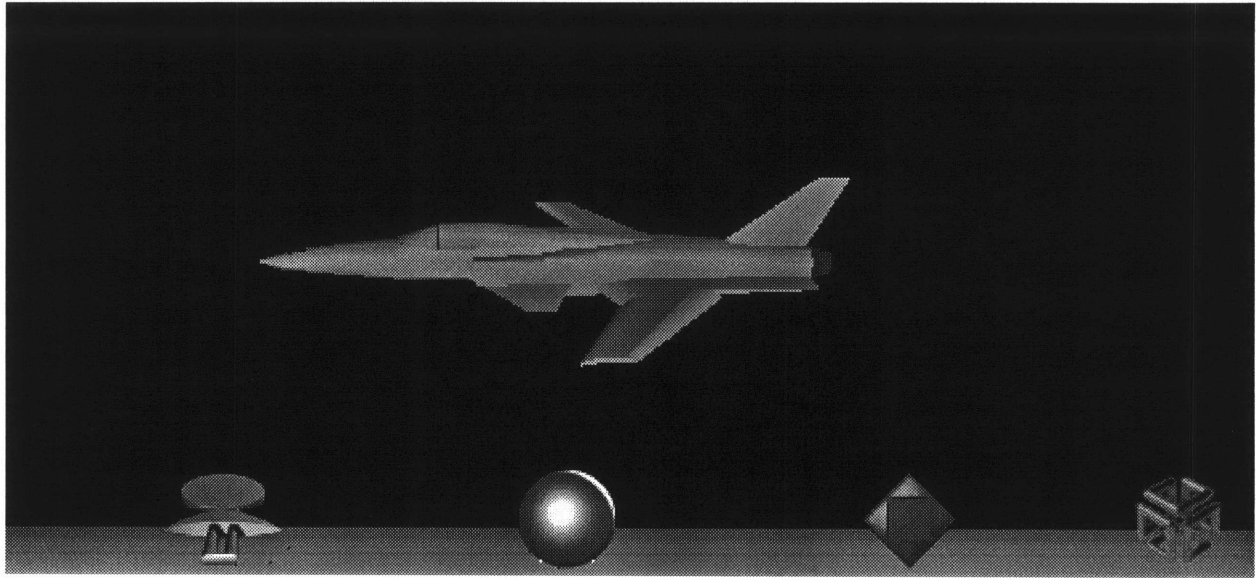

Figure 3.5c: VRML User Interface This is an example of the interface using Cosmo Player from Silicon Graphics, Inc.

4.0 Case Studies

Advantages to the system just described include the ability to communicate with others while viewing the same three dimensional spatial representation and the ability to show a person your point of view of this representation even though there may be great physical separation. Two examples where such a collaborative inspection tool may be useful will be described in section 4.1 and 4.2. The first example involves an architect in California and a structural engineer in New York. The engineer has received the AutoCAD drawing files for a new building they are designing, but has some points of concern she would like to address with the architect. The second example involves a newly constructed laboratory building housing dangerous chemicals. The local fire department has received the AutoCAD plans for the building and the fire chief wishes all firefighters to experience a virtual training session with the scenario being that a fire has just been discovered in this building.

4.1 Collaborative Architectural Study

This scenario begins with an architect and a structural engineer, both working on the design of a new office building to be constructed in New York City. The architect is located in San Francisco, California, and engineer works from an office in downtown Manhattan, New York. The engineer has just received the AutoCAD drawings for the new building from the architect and she has some questions about the design.

Because they are located in separate cities and busy schedules do not allow for travel, they initially try to discuss the design over the phone. To her dismay, the engineer cannot

accurately describe for the architect the location or nature of the problem. If only she could point to the location on the drawing itself and have him **see** the area that she means.

This matter must be cleared up before any structural analysis can take place. Deadlines need to be met, so they can not bother with sending drawings back and forth from coast to coast. They need a real-time method for collaborating on this drawing. This is a case in which the collaborative inspection tool can help.

The engineer happens to be running a web server from the desk-top computer in her office. She copies the necessary Java programs to her hard drive. The client side applet is located in her web server directory and the server program may be installed in any location of her choice. She has a Java enabled web browser with a VRML interpreter plug-in installed. The architect informs her that he has the same. The engineer converts the AutoCAD file to VRML (see figure 4.1a below) and adds the appropriate Viewpoint and ProximitySensor nodes[18]. The VRML filename and applet name are embedded into an HTML document[19] which is located in her web server directory. She starts the server program by typing the execution command at the command line prompt. The architect does not need any of these files, only a connection to internet and permission to access the engineer's web server.

Figure 4.1a: Conversion from AutoCAD to VRML One way to convert an AutoCAD file to VRML is to output the drawing to a 3DStudio file, convert that to Open Inventor using a conversion utility from Silicon Graphics, Inc. (SGI), and finally convert that output to VRML with another utility also from SGI.

---

[18] See Appendix C for these nodes.
[19] See Appendix D for a sample of this HTML document.

Once this is set up (it is estimated to take about one hour to do so), they schedule a time to hold a meeting on the internet. At this designated time, they both open the proper URL[20] in their web browser windows. The VRML file is loaded on both computers and connections are made to the Java server program.

The engineer notifies the architect that she will show him the area of concern by taking control of the viewer and navigating the virtual world. (Chatting is done by typing a message in the text field and sending it through the server. Control is taken by clicking on the appropriate button.) Once she has taken control of the viewer, she navigates the three dimensional environment with the control panel of the viewer. At the same time that movement is occurring on her screen, this movement is echoed on the screen of the architect. She pans in to a column section and expresses the reason for her concern in the chat text area.

The architect responds to her concerns and in an effort to clear up the matter, he takes control of the viewer and navigates to another section of the building which has an effect on the potential problem area. The engineer is able to see this as the motion in her VRML viewer is controlled by the architect's movements. Once the meeting has completed, they say goodbye and exit the program by closing their browser windows.

---

[20] Universal Resource Locator.

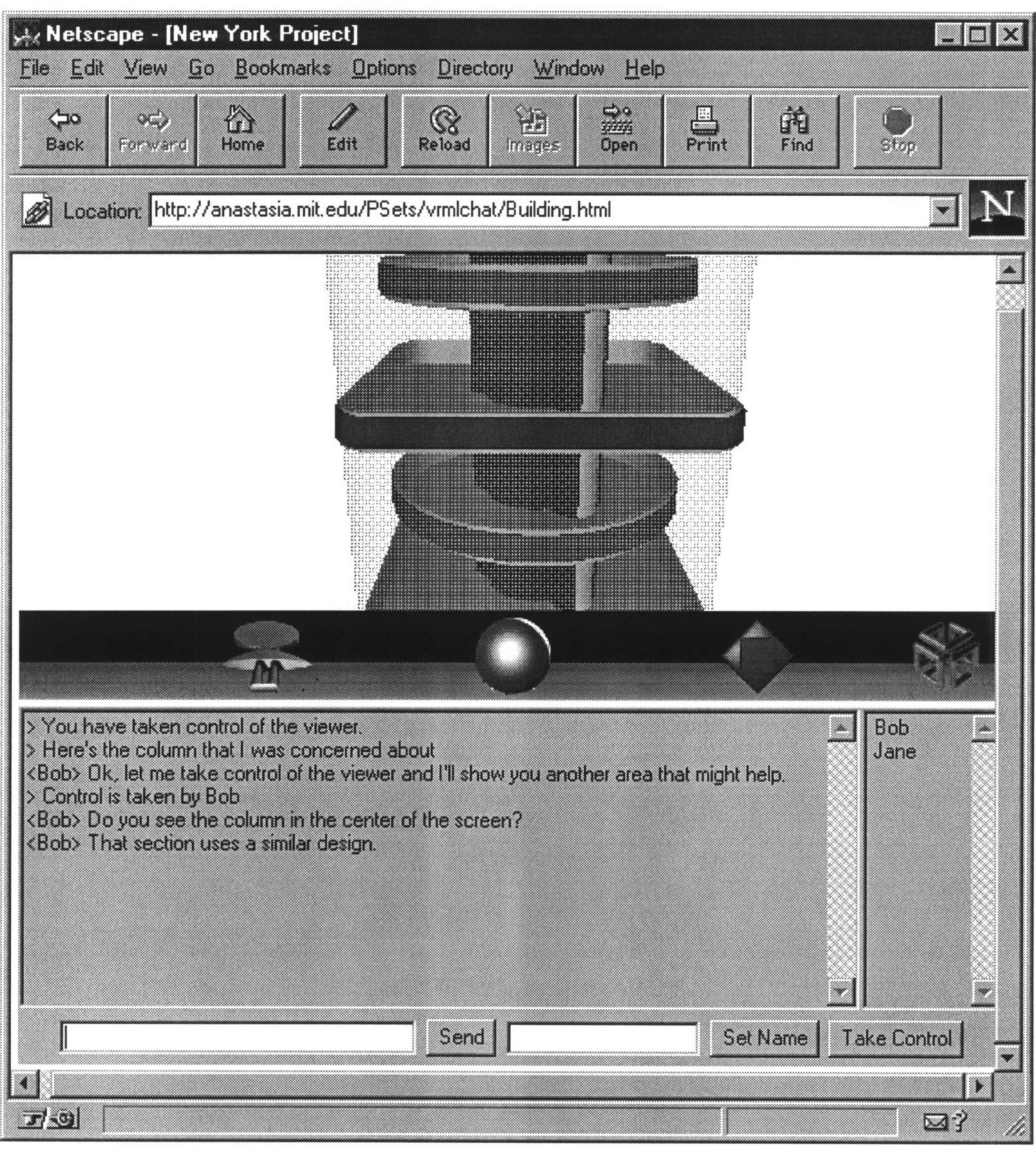

Figure 4.1b: Screen Shot of Building Design User Interface This is what users might see when running the collaborative inspection tool for examining the design of a building.

4.2 Fire Safety Training

In this second scenario, MIT has just constructed a new laboratory building for the chemistry department. In order to aid the efforts of the local fire department, they give a copy of the AutoCAD plans to the fire chief. There are many areas in the building where dangerous, flammable chemicals are stored and these are clearly designated in the drawing files.

Knowing that special treatment will be required should a fire ever develop in this building, the fire chief holds a virtual training drill using the new collaborative inspection tool. The fire department happens to have a computer cluster and a web server, so the fire chief installs all the appropriate Java programs and starts the server. As in the case of the architect and the engineer, he converts the drawings to VRML and creates an HTML document which includes both the virtual world and the Java applet program.

Once this is complete, he has all of the fire fighters come down to the computer cluster and open their web browsers to the HTML page. Then he explains the scenario that a fire has been discovered in one of the laboratory rooms. All fire fighters watch their screens as he traverses the virtual building in search of trapped students and the source of the fire. As he enters an area where hazardous chemicals are stored, he can explain the special precautions that must be taken to avoid fatal injury.

As the fire fighters watch, they have the option of conversing over the chat program to ask questions about what they should do in this scenario or what they can expect when dealing with certain chemicals. Even if all fire fighters are sitting in the same room during this virtual drill (in some cases, all computers may not be located in the same room), it may be preferred to use the chat program over simply speaking aloud, in order to facilitate a more orderly question

and answers session and also to provide a visual memory for this important information to be recalled should such an emergency arise.

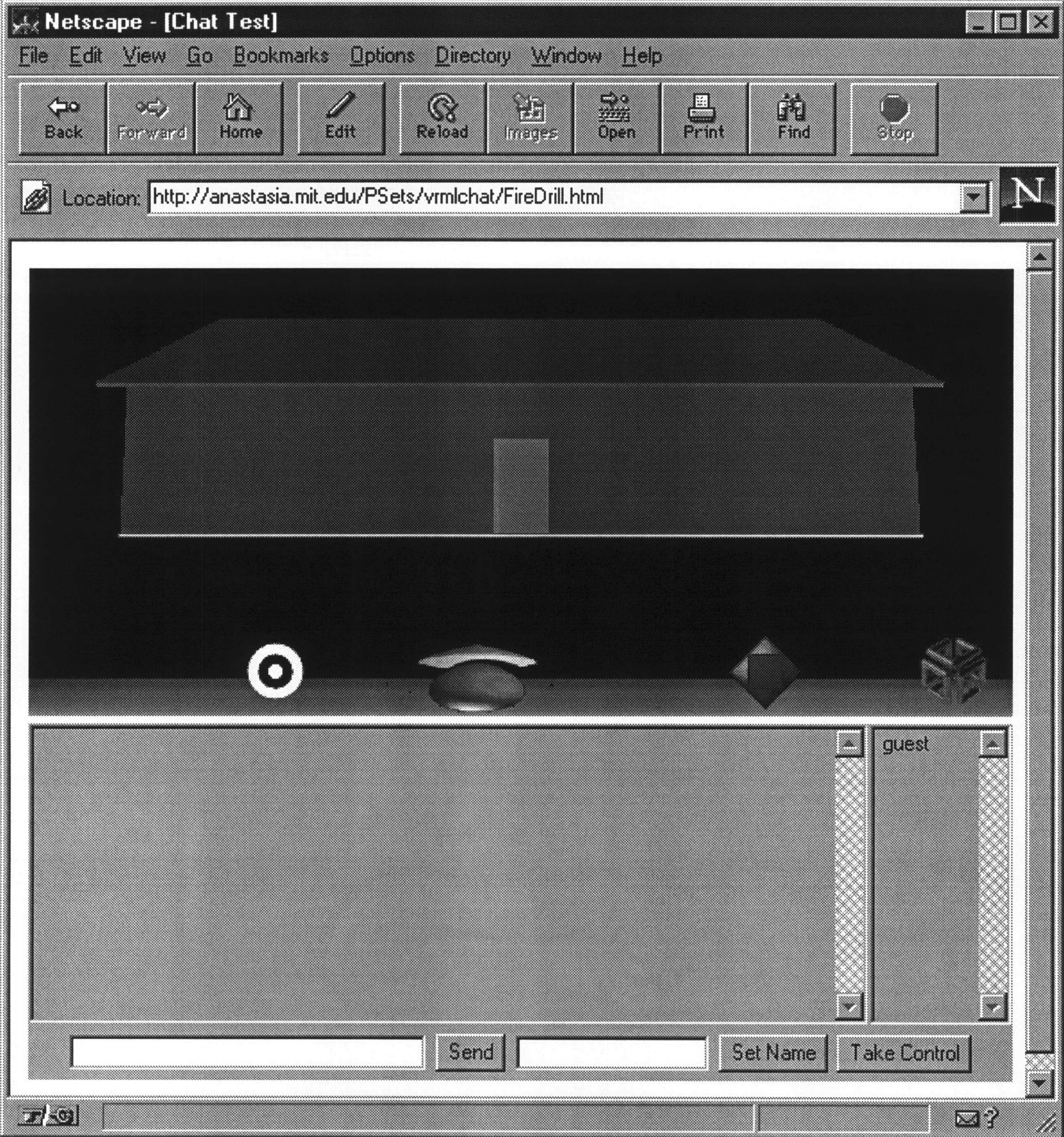

Figure 4.2a: Screen Shot of Fire Safety Training User Interface This is what users would see when running the collaborative inspection tool for a virtual Fire Safety Training exercise.

5.0 Future Development

Although the collaborative inspection tool as it stands offers many useful advantages for this process, there are is still much room for improvement. Section 5.1 will discuss some of these possible improvements and give suggestions for their implementation. In section 5.2, different variations on the program will be discussed, dealing with issues of customizing the program for specific applications.

5.1 Improvements

One possible improvement to the collaborative inspection tool would be to have a remote method for starting and stopping the Java server. Because it is not necessary that anyone be using the tool from the same machine the server is running on, it is possible that one might find themselves in another location (possibly a business trip or visit to a client's office) and wish to run the collaborative tool. Unless the server was started before you left the machine, or someone is available to run the startup command for you, you will not be able to use the tool. It would be possible to write a CGI program (perhaps in PERL) so that any person who could access the page through a web browser and supply the correct username and password (if you wished to protect it), could start and stop the Java server remotely.

Another improvement to the collaborative inspection tool that can be made is to add a function that allows the users to dynamically create VRML "arrows" in the virtual world to point out specific areas of interest while they are discussing that area's various aspects. Using the EAI **createVrmlFromString** function in a Java applet, this method could easily be implemented.

In an effort to make traversing the virtual world a little easier for users, a function might be added to control movement with the keyboard. By using the **keyUp** function in the Applet base class, you could translate every press of the "up arrow" key into positive movement in the z-direction. The right and left arrow keys could turn the user's viewpoint to the right and left, respectively, and so on.

An additional improvement would be to add the capability to open any VRML file into the collaborative inspection tool without first having to add special nodes ZERO and MOTIONTRACKER. By having a generic VRML file which contains these two nodes (in addition to a special **Inline** node) and embedding it into the HTML document, you could write a method in the Java applet that creates an instance of this file's Inline node. Once the applet is told to load a specific VRML file, the URL of that file will be sent to the corresponding field on the Inline node and the new virtual world will be added.

5.2 Variations on Design and Use

In section 4, two very different scenarios were given for instances in which the collaborative inspection tool would be useful. This gives rise to the idea that it may be beneficial to create multiple versions of this tool, each customized with a particular purpose in mind. Below are examples of such variations.

It is possible to embed more than one VRML file into an HTML document. If the number of connections was limited to between two or four, you could allow the users to see **each** other user's viewpoint as well as their own. This would be useful in a design inspection process where each collaborator may have a unique point of interest they wish to show to the other users.

Also for the case of design, it would be useful to have a version of the tool where information about each object may be linked to its visual representation. Then, when a user clicks on that object in the VRML window, the information can be displayed in the chat text window. Such information could be material properties, dimensions, or weight.

In the case of the fire fighters' virtual training drill, a variation of the tool could have been made to allow users to see the **position** of other users, instead of just looking at the world through one user's eyes. With this variation, they would all be able to run the drill simultaneously and familiarize themselves with where other fire fighters might be inside the building. This variation would allow the tool to become a more realistic simulation of the real world.

6.0 Conclusion

In today's competitive business market, long-distance collaboration is key to pooling the best resources the industry has to offer. With this tool, three dimensional representations are linked to text based network communication. People no longer need to be in the same office to work on the same project. Ideas, thoughts, and decisions can now be shared across internet lines, spanning thousands of miles in nanoseconds of time.

The collaborative inspection tool just described in the text allows users to link together to share ideas, concepts, and concerns about three dimensional objects. Implemented with Java and VRML, it spans the internet with powerful networking capabilities and graphical rendering. The application and expandability of this tool are as limitless as the imagination.

References

1. G. Cornell, C. S. Horstmann, Core JAVA, SunSoft Press: A Prentice Hall Title, Mountain View, CA, 1996.
2. A. Scott, *The Marriage of VRML and Java*, Aereal, Inc., Location: http://www.vrmlsite.com/sep96/spotlight/javavrml/javavrml.html, Visited on 5 May 1997.
3. C. Marrin, *Proposal for a VRML 2.0 Informative Annex: External Authoring Interface Reference*, Silicon Graphics, Inc., Location: http://vrml.sgi.com/moving-worlds/spec/ExternalInterface.html, Visited on 5 May 1997.

Appendix A: Server Code

The following is all source code for the four server classes: MCserver, MCclientThread, MCsocketArray, and MCarray. For these four classes, all code was written by Manuel Perez (M.Eng. EECS MIT '97)

MCserver.java

```
import java.lang.Thread;
import java.io.*;
import java.net.*;
import MCsocketArray;
import MCclientThread;

/**
 * MCserver
 * -------
 * MiniChat Server
 *
 * @version 0.2
 * @date    September 5, 1996
 * @author  Manuel Perez
 */
public class MCserver {

  /**
   * signals the correct start of a server
   */
  public boolean started;

  /**
   * The server socket
   */
  public ServerSocket s;

  /**
   * Array of client sockets
   */
  public MCsocketArray sa;

  /**
   * Array of usersnames
   */
  public String[] users;

  /**
   * Constructs a new MicroChat server. Parses the input
   * arguments for port number and connection number. Sets
   * the internal variable started to true if the server
   * has started correctly
```

```
 *
 * @param args   comamnd line arguments
 */
public MCserver(String args[]) {
  started = parseInputs(args);
  if(started) {
    try {
      s = new ServerSocket(port);
      sa = new MCsocketArray(connections);
      users = new String[connections];
      System.out.println
        ("Opening server on port " +   port + ", Accepting " +
        connections + " connections.");
    }
    catch (Exception e) {
      System.out.println(e);
      started = false;
    }
  }
}

/**
 * Parses the input arguments for port number and connection
 * number.
 *
 * @param args   comamnd line arguments
 */
private boolean parseInputs(String[] args) {
  port = defaultPort;
  connections = defaultConnections;
  int size = args.length;
  for(int i=0;i<size;i++) {
    if(args[i].equals("-port")) {
      if(i+1<size) {
        try {
          port = Integer.parseInt(args[i+1]);
          i++;
        }
        catch (Exception e) {
          System.out.println("Bad port number");
          return(false);
        }
      }
      else {
        System.out.println("Must select port number");
        return(false);
      }
    }
    if(args[i].equals("-c")) {
      if(i+1<size) {
        try {
          connections = Integer.parseInt(args[i+1]);
          i++;
        }
        catch (Exception e) {
```

```
            System.out.println("Bad connection number");
            return(false);
          }
        }
        else {
          System.out.println("Must select connection number");
          return(false);
        }
      }
    }
    return(true);
  }

  public static void main(String[] args) {
    try {
      MCserver server =  new MCserver(args);
      if(server.started) {
        for(;;) {
          Socket incoming = server.s.accept();
            new MCclientThread(incoming, server.sa, server.users).start();
        }
      }
      else {
        System.out.println("Can not start server.");
      }
    }
    catch (Exception e) {
      System.out.println(e);
    }
  }

  private int port;
  private int connections;
  private static int defaultPort = 9992;
  private static int defaultConnections = 10;
}
```

MCclientThread.java

```
import java.lang.Thread;
import java.io.*;
import java.net.*;
import MCsocketArray;
/**
 * MCclientThread
 * -------------
 * Handles client
 *
 * @version 0.2
 * @date    September 5, 1996
```

```
 * @author  Manuel Perez
 */
public class MCclientThread extends Thread {
  /**
   * Constructs a new MCclientThread.
   *
   * @param soc   Client Socket
   * @param sa    Array of all sockets
   */
  public MCclientThread(Socket soc, MCsocketArray sa, String[] use) {
    cSocket = soc;
    socketArray = sa;
    loop = true;
    UID = GID++;
    name = "guest";
    try {
      sa.addSocket(soc);
      users = use;
      for(int i=0;i<use.length;i++) {
        if(use[i]==null) {
          use[i]=name;
          break;
        }
      }
    }
    catch (Exception e) {
      //handles too many
    }
  }

  /**
   * Returns a unique identifier of the thread
   *
   * @returns   Unique Identifier
   */
  public long getUID() {
    return(UID);
  }

  /**
   * Sends a message to all the clients, except the sender
   *
   * @param message   the message to be sent
   */
  public void sendAll(String message) {
    for(int i=0;i<socketArray.size();i++) {
      if(socketArray.getSocket(i)!=cSocket) {
        try {
          DataOutputStream out = new DataOutputStream
            (socketArray.getSocket(i).getOutputStream());
          out.writeUTF("<" + name + "> " + message + "\n");
        }
        catch (Exception e) {
        }
      }
```

```
    }
  }

  /**
   * Sends a user list to all clients
   */
  public void sendUserList() {
    String message = "";
    for(int i=0;i<users.length;i++) {
      if(users[i]!=null) {
        message = message + " " + users[i] + "\n";
      }
    }
    for(int i=0;i<socketArray.size();i++) {
      try {
        DataOutputStream out = new DataOutputStream
          (socketArray.getSocket(i).getOutputStream());
        out.writeUTF("USERLIST " + message + "\n");
      }
      catch (Exception e) {
      }
    }
  }

  /**
   * Start of thread. waits for client inputs and then
   * relays this information to all other clients
   */
  public void run() {
    try {
      DataInputStream in =
        new DataInputStream(cSocket.getInputStream());
      sendUserList();
      while(loop) {
        String message = in.readUTF();
        if(message!=null) {
          if(!parseMessage(message)) {
            sendAll(message);
          }
        }
        else {
          loop = false;
        }
      }
        cSocket.close();
    }
    catch (Exception e) {
    }
    System.out.println("Lost client");
    socketArray.delSocket(cSocket);
    for(int i=0;i<users.length;i++) {
      if(users[i]!=null) {
        if(users[i].equals(name)) {
          users[i]=null;
          break;
```

```
      }
    }
  }
  sendUserList();
}

private boolean parseMessage(String message) {
  if(message.indexOf("SETNAME") >= 0) {
    for(int i=0;i<users.length;i++) {
      if(users[i]!=null) {
        if(users[i].equals(name)) {
          users[i]=null;
          break;
        }
      }
    }
    name = message.substring(message.indexOf("SETNAME")+8);
    for(int i=0;i<users.length;i++) {
      if(users[i]==null) {
        users[i]=name;
        break;
      }
    }
    sendUserList();
    return(true);
  }
  else if(message.indexOf("USERLIST") >= 0) {
    return(true);
  }
  return(false);
}

private static long GID = 0;
private long UID;
private Socket cSocket;
private MCsocketArray socketArray;
private boolean loop;
private String name;
private String[] users;
}
```

MCsocketArray.java

```
import java.net.*;
import MCarray;
/**
 * MCsocketArray
 * -------------
 * Maintains an array of sockets
 *
 * @version 0.3
 * @date    September 5, 1996
```

```
 * @author  Manuel Perez
 */
public class MCsocketArray {
  /**
   * Constructs a new MCsocketArray
   *
   * @param size   size of the array
   */
  MCsocketArray(int size) {
    s = new MCarray(size);
  }

  /**
   * Trys to add a new socket to the MCsocketArray.
   *
   * @param soc   the socket to be added
   * @exception   Array_Full, thrown if the array
   *            is full
   */
  public void addSocket(Socket soc) throws Exception{
    try {
      s.add((Object)soc);
    }
    catch (Exception e) {
      throw e;
    }
  }

  /**
   * Deletes the input socket if it is in the
   * array
   *
   * @param soc   the socket to be deleted
   */
  public void delSocket(Socket soc) {
    s.del((Object)soc);
  }

  /**
   * Looks up a socket by index and returns that
   * socket if found.
   *
   * @param index   index of the requested socket
   * @returns       the requested socket, or null if
   *             the socket was not found
   */
   public Socket getSocket(int index) {
    return((Socket)s.get(index));
  }

  /**
   * Returns the size of the MCsocketArray
   *
   * @returns   size of the array
   */
```

```
  public int size() {
    return(s.size());
  }

  private MCarray s;
}
```

MCarray.java

```
/**
 * MCArray
 * -------
 * Maintains an array
 *
 * @version 0.1
 * @date    September 5, 1996
 * @author  Manuel Perez
 */
public class MCarray  {
  /**
   * Constructs a new MCarray
   *
   * @param size   size of the array
   */
  MCarray(int size) {
    a = new Object[size];
    num = 0;
    max = size;
  }

  /**
   * Trys to add a new element to the MCarray.
   *
   * @param soc   the elementto be added
   * @exception   Array_Full, thrown if the array
   *            is full
   */
  public void add(Object obj) throws Exception{
    if(num==max) {
      throw new Exception("Array_Full");
    }
    else {
      a[num] = obj;
      num++;
    }
  }

  /**
   * Deletes the input element if it is in the
   * array
   *
   * @param soc   the element to be deleted
```

```
 */
public void del(Object obj) {
  int i=0;
  for(i=0;i<num;i++) {
    if(a[i]==obj) {
      break;
    }
  }
  if(i!=num) {
    a[i] = a[num-1];
    a[num-1] = null;
    num--;
  }
}

/**
 * Looks up a element by index and returns that
 * element if found.
 *
 * @param index   index of the requested element
 * @returns       the requested element, or null if
 *                the element was not found
 */
public Object get(int index) {
  if(index>=num) {
    return(null);
  }
  else {
    return(a[index]);
  }
}

/**
 * Returns the size of the array
 *
 * @returns   size of the array
 */
public int size() {
  return(num);
}

private Object[] a;
private int num;
private int max;
}
```

Appendix B: Java Client Code

The following is all source code for the three client side Java classes: MCclient, MCreceive, and MCsend. Classes MCclient and MCreceive were modified by Carrie Morton to suite the purpose of a VRML collaborative inspection tool. The originals programs were created by Manuel Perez.

MCclient.java

```
import java.lang.Thread;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.applet.*;
import java.util.*;
import netscape.javascript.JSObject;
import netscape.plugin.Plugin;
import netscape.javascript.JSException;
import vrml.external.field.*;
import vrml.external.exception.*;
import vrml.external.Node;
import vrml.external.Browser;
import vrml.node.*;
import vrml.field.*;
import MCsend;
import MCreceive;

/**
 * MCclient(Application)
 * ---------------------
 *   A MiniChat Client Window.
 *
 * @args      -host <server>
 *            -port <server port>
 *
 * @version   0.5
 * @created   September 5, 1996
 * @modified  September 9, 1996
 * @author    Manuel Perez
 *
 * @modifications
 *   -added comments(september 8, 1996)
 *   -fixed null string bug
 *   -added buffer size
 */

// Modified by Carrie Morton on May 5, 1997
// Modifications include addition of VRML interaction in the init function,
// the addition of functions callback, moveWorld, and ChangeControl
// Modifications to the user interface also made.
```

```
// Class SetNameDialog was deleted.

public class MCclient extends Applet implements EventOutObserver {

  public void init() {


          // This is the EAI stuff (from example at SGI)
          JSObject win = JSObject.getWindow(this);
    JSObject doc = (JSObject) win.getMember("document");
    JSObject embeds = (JSObject) doc.getMember("embeds");
    browser = (Browser) embeds.getSlot(0);


          // attempt to catch exceptions
          while(!thingsok) {  // need this to get around the problem of java possibly loading before vrml
          try {
     view = browser.getNode("ZERO");
                  sensor = browser.getNode("MOTIONTRACKER");
                  position_changed = (EventOutSFVec3f) sensor.getEventOut("position_changed");
                  position_changed.advise(this, null);                    //this sets up the "callback"
                  orientation_changed = (EventOutSFRotation) sensor.getEventOut("orientation_changed");
                  new_position = (EventInSFVec3f) view.getEventIn("position");
           new_orientation = (EventInSFRotation) view.getEventIn("orientation");

                  thingsok = true;
    }
    catch (InvalidNodeException e) {
     System.out.println("PROBLEMS!: " + e);
                  thingsok = false;
    }
          } // end of while


    /*Initialize internal variables*/
    message = "";
    sendmessage = null;

    /*Window initilaization*/
    setLayout(new BorderLayout());

    /*Send Box and set name button*/
    Panel p = new Panel();
    input = new TextField(30);
    p.add(input);
          p.add(new Button("Send"));
          namebox = new TextField(15);
          p.add(namebox);
    p.add(new Button("Set Name"));
          p.add(new Button("Take Control"));
          Control = 0;
    add("South", p);

    /*Main Text Box*/
    ta = new TextArea(8,10);
```

```
  ta.setEditable(false);
  add("Center", ta);

  /*UserBox*/
  users = new TextArea(8,10);
  users.setEditable(false);
  add("East", users);

  /*Final initialization*/
  buffersize=0;
  String host = getParameter("host");
  int port = Integer.parseInt(getParameter("port"));
  receiver = new MCreceive(host, port, this);
  sender = new MCsend(receiver.cSocket, this);
  receiver.start();
  sender.start();
}
//end of init() function

// callback to the EventOutObserver class
public void callback(EventOut event, double time, Object userData) {
   //System.out.println("Touchdown");  was used for debugging

          float pos[] = new float[3];                    // maybe declare these var's outside function
          float orient[] = new float[4];        // so as to save mem.
          pos = position_changed.getValue();
          orient = orientation_changed.getValue();
          if (Control == 1) {
          sendmessage = "MOVEMENT " + pos[0] + " " + pos[1] + " " + pos[2] + " " + orient[0] + " " + orient[1]
+ " " + orient[2] + " " + orient[3] + " \n";
          }

}

/**
 * Adds a new message to the main text box
 *
 * @param str   message string
 */
public void setMessage(String str) {
  message = str;
  if(maxbuffer!=-1) {
        buffersize++;
        if(buffersize>=maxbuffer){
          int index = ta.getText().indexOf("\n");
          if(index==-1) index = ta.getText().indexOf("\r");
          ta.setText(ta.getText().substring(index+1));
          buffersize--;
        }
  }
  ta.appendText(message);
}

public void moveWorld(String str) {
```

```
		message = str;
		float[] position = new float[3];
		float[] orientation = new float[4];
		int i = str.indexOf(" ");

		String rest = str.substring(0,str.length());
		i = rest.indexOf(" ");
		position[0] = (Float.valueOf(rest.substring(0, i-1))).floatValue();

		rest = rest.substring(i+1,rest.length());
		i = rest.indexOf(" ");
		position[1] = (Float.valueOf(rest.substring(0, i-1))).floatValue();

		rest = rest.substring(i+1,rest.length());
		i = rest.indexOf(" ");
		position[2] = (Float.valueOf(rest.substring(0, i-1))).floatValue();

		rest = rest.substring(i+1,rest.length());
		i = rest.indexOf(" ");
		orientation[0] = (Float.valueOf(rest.substring(0, i-1))).floatValue();

		rest = rest.substring(i+1,rest.length());
		i = rest.indexOf(" ");
		orientation[1] = (Float.valueOf(rest.substring(0, i-1))).floatValue();

		rest = rest.substring(i+1,rest.length());
		i = rest.indexOf(" ");
		orientation[2] = (Float.valueOf(rest.substring(0, i-1))).floatValue();

		rest = rest.substring(i+1,rest.length());
		i = rest.indexOf(" ");
		orientation[3] = (Float.valueOf(rest.substring(0, i-1))).floatValue();

		System.out.println(position[0] + " " + position[1] + " " + position[2] + " " + orientation[0] + " " +
orientation[1] + " " + orientation[2] + " " + orientation[3] + " \n");

		new_position.setValue(position);
		new_orientation.setValue(orientation);

				//this is where i send position and orientation
				//vectors to the VRML world

	}

	public void ChangeControl (String str) {
		String controller = new String();
		Control = 0;
		int i = str.indexOf("TAKINGCONTROL");
		controller = str.substring(1, i-2);
		setMessage("> Control is taken by " + controller + "\n");
	}

	/**
	 * Updates the user list
	 *
```

```
 * @param str   user list
 */
public void setUsers(String str) {
  message = str;
  users.setText(message);
}

/**
 * Returns a new send message. Returns null if no
 * new message is present
 *
 * @return   new message if it exist, else null
 */
public String getSendMessage() {
  return(sendmessage);
}

/**
 * Resets the send message
 */
public void resetSendMessage() {
  sendmessage=null;
}

public boolean handleEvent(Event e) {
  if (e.id == Event.WINDOW_DESTROY)
    System.exit(0);
  return(super.handleEvent(e));
}

public void paint(Graphics g) {
}

public boolean keyUp(Event evt, int key) {
  if(key==10) /*handles <enter> press*/ {
    sendmessage = input.getText();
    setMessage("> "+ input.getText() + "\n");
          input.setText("");
  }
  return(true);
}

public boolean action(Event evt, Object arg) {
  if(evt.target instanceof MenuItem) {
    if(arg.equals("Quit")) {
      System.exit(0);
    }
  }
  else if(evt.target instanceof Button) {
    if(arg.equals("Set Name")) {
      sendmessage = "SETNAME " + namebox.getText();
                    namebox.setText("");
    }
              if(arg.equals("Send")) {
                    sendmessage = input.getText();
```

```
        setMessage("> "+ input.getText() + "\n");
                input.setText("");
      }
                    if(arg.equals("Take Control")) {
                            Control = 1;
                            sendmessage = "TAKINGCONTROL";
                            setMessage("> You have taken control of the viewer. \n");
                    }
    }
    else return(false);
    return(true);
  }

  /*variables*/
  private String message;          /*received messages*/
  private TextArea ta;             /*main text area*/
  private TextArea users;          /*user list*/
  private TextField input;         /*send message input box*/
  private TextField namebox;       /*set name input box*/
  public  String sendmessage;      /*sent messages*/
  private MCreceive receiver;      /*receive thread*/
  private MCsend sender;           /*sender thread*/
  private int buffersize;          /*main text box buffer size*/
  private static int maxbuffer = -1; /*maximum size of the buffer*/
  private int Control;             //0=not in control; 1=in control of motion
  Browser browser;
  Node view = null;
  Node sensor = null;
  EventOutSFTime touchTime = null;
  EventOutSFVec3f position_changed;
  EventOutSFRotation orientation_changed;
  EventInSFVec3f new_position;
  EventInSFRotation new_orientation;
  private boolean thingsok = false;  //for loading java before vrml


}
```

MCreceive.java

```
import java.lang.Thread;
import java.io.*;
import java.net.*;
import java.awt.*;
import MCclient;

/**
 * MCreceive
 * ---------
 *   A thread that handles client receive messages.
 *
 * @version   0.5
 * @created   September 5, 1996
```

```
 * @modified   September 8, 1996
 * @author     Manuel Perez
 *
 * @modifications
 *   -added comments(september 8, 1996)
 *
 */

// Modified by Carrie Morton on May 5, 1997
// Modification include addition of MOVEMENT and TAKINGCONTROL
// cases in the parseMessage function necessary for VRML interaction

public class MCreceive extends Thread {

  /**
   * Constructs a new MCreceive thread. This method
   * opens a new socket.
   *
   * @param host    the server host
   * @param port    the port number on the server
   * @param w       the main program
   */
  public MCreceive(String host, int port, MCclient w) {
    W = w;  /*sets relation between thread and client program*/
    try {
      /*attempts to connect to server*/
      cSocket = new Socket(host, port);
      loop = true;
    }
    catch (Exception e) {
      /*handles failed connect*/
      loop = false;
    }
  }

  /**
   * Main program loop for the thread(standard)
   */
  public void run() {
    try {
      /*attempts to open up an input stream*/
      DataInputStream in =
        new DataInputStream(cSocket.getInputStream());

      /*loops until loop=false*/
      while(loop) {
        String message = in.readUTF();  /*get new input*/
        /*checks for special message*/
        if(!parseMessage(message)) {
          /*if no special message the message is sent
           *to the main program to be posted
           */
          W.setMessage(message);
        }
        W.repaint();  /*tells main program to redraw the screen*/
```

```
      }
    }
    catch (Exception e) {
      /*handles exception by doing nothing*/
    }
  }

  /**
   * Scans input for special commands and runs these
   * commands. Returns true if a special command has
   * been parsed/
   *
   * @param message    the message to be examined
   * @returns          true, if command found, else false
   */
  private boolean parseMessage(String message) {
    if(message.indexOf("USERLIST") >= 0) {
      /*handles USERLIST update*/
      W.setUsers(message.substring(message.indexOf("USERLIST")+9));
      return(true);
    }
        else if(message.indexOf("MOVEMENT") >= 0) {
      /*handles movement in VRML world*/
      W.moveWorld(message.substring(message.indexOf("MOVEMENT")+9));
      return(true);
    }
        else if(message.indexOf("TAKINGCONTROL") >= 0) {
                /*handles a user taking control of the VRML viewer*/
                W.ChangeControl(message);
                return(true);
        }

    return(false);
  }

  /*variables*/
  public Socket cSocket;      /*The communication socket connection*/
  private MCclient W;         /*Reference to the main client program*/
  private boolean loop;       /*Continuation of main loop boolean*/
}
```

MCsend.java

```
import java.lang.Thread;
import java.io.*;
import java.net.*;
import java.awt.*;
import MCclient;

/**
 * MCsend
 * ------
```

```
 *   A thread that handles client send messages.
 *
 * @version    0.5
 * @created    September 5, 1996
 * @modified   September 8, 1996
 * @author     Manuel Perez
 *
 * @modifications
 *    -added comments(september 8, 1996)
 *
 */
public class MCsend extends Thread{

  /**
   * Constructs a new MCsend thread. This method
   * requires a precreated socket.
   *
   * @param host    the server host
   * @param port    the port number on the server
   * @param w       the main program
   */
  public MCsend(Socket soc, MCclient w) {
    W = w;  /*sets relation between thread and client program*/
    cSocket = soc;  /*sets relation with a socket*/
    loop = true;
  }

  /**
   * Main program loop for the thread(standard)
   */
  public void run() {
    try {
      /*attempts to open up an output stream*/
      DataOutputStream out =
        new DataOutputStream(cSocket.getOutputStream());

      /*loops until loop=false*/
      while(loop) {
        /*loops until a message is readied by the main program*/
        while(W.getSendMessage()==null) {
          sleep(5);  /*sleeps for 5ms*/
        }
        String message = W.getSendMessage();
        W.resetSendMessage();
        out.writeUTF(message);
        W.repaint();  /*tells main program to redraw the screen*/
      }
    }
    catch (Exception e) {
      /*handles exception by doing nothing*/
    }
  }

  /*variables*/
  public Socket cSocket;        /*The communication socket connection*/
```

```
  private MCclient W;          /*Reference to the main client program*/
  private boolean loop;        /*Continuation of main loop boolean*/
}
```

Appendix C: VRML Client Code

The following two nodes, ZERO and MOTIONTRACKER, must be added to any VRML file in order to use it with this program.

```
DEF ZERO Viewpoint {
            position      -0.91971 1.11424 25.5731
            description  "Home"
        },

DEF MOTIONTRACKER ProximitySensor {
 center 0 0 0
 size 1000 1000 1000
},
```

Appendix D: Sample Files from Case Studies

Following are sample VRML and HTML files for the two case studies discussed in section 4.1 and 4.2 of this text. Some sections of the VRML files have been omitted for brevity.

HTML file for case of Architect and Engineer

```
<html>
<head>
<title>New York Project</title>
</head>

<center>
<embed src="../../final2.wrl" border=0 height="250" width="550">
</center>

<center>
<applet code="MCclient.class" mayscript height="200" width="550">
<PARAM NAME="host" VALUE="anastasia.mit.edu">
<PARAM NAME="port" VALUE="9992">
</applet>
</center>

</html>
```

VRML file for case of Architect and Engineer

```
#VRML V2.0 utf8

NavigationInfo {
  headlight   TRUE
  type        "EXAMINE"
  avatarSize  .5
}

DEF ZERO Viewpoint {
        position -0.91971 1.11424 25.5731
        description         "Home"
},

DEF MOTIONTRACKER ProximitySensor {
 center 0 0 0
 size 1000 1000 1000
},

        Shape {
          appearance
                Appearance {
```

```
                material
                        Material {
                          ambientIntensity        0.2
                          diffuseColor   0.5 0.5 0.7
                          specularColor  0 0 0
                          emissiveColor  0 0 0
                          shininess      0.2
                        }
              }
        geometry
              IndexedFaceSet {
                coord
                        Coordinate {
                          point  [ AutoCAD output omitted here ]
                        }
                normal
                        Normal {
                          vector [ AutoCAD output omitted here ]
                        }
                color
                        Color {
                          color  [ 0.5 0.5 0.7,
                                   0.7 0.7 0.8,
                                   0 0 0,
                                   0.8 0.5 0.2,
                                   0.8 0.2 0.2 ]
                        }
                colorPerVertex FALSE
                creaseAngle    0.5
                coordIndex     [ AutoCAD output omitted here ]
                colorIndex     [ AutoCAD output omitted here ]
                normalIndex    [AutoCAD output omitted here]
              }
        }
    ]
}
```

HTML file for case of Fire Department

```
<html>
<head>
<title>Chat Test</title>
</head>

<center>
<embed src="../museum4.wrl" border=0 height="250" width="550">
</center>

<center>
<applet code="MCclient.class" mayscript height="200" width="550">
<PARAM NAME="host" VALUE="anastasia.mit.edu">
<PARAM NAME="port" VALUE="9992">
</applet>
```

</center>

</html>

VRML file for case of Fire Department

```
#VRML V2.0 utf8

DEF SCENE_VIEWS Transform {
  children [
    DEF Cameras Group {
      children [
        DEF ZERO Viewpoint {
          position  0 5 100
description "zero"
        }
            DEF first Viewpoint {
          position  0 3 10
description "first"
        }] } ] }

DEF MOTIONTRACKER ProximitySensor {
 center 0 0 0
 size 1000 1000 1000
}

Transform {
  children [
        Transform {
          children [
                Transform {
                  children [
                        DEF INTDOORS Transform {
                          children [
                                Transform {
                                  children [
                                        Shape {
                                          appearance
                                                Appearance {
                                                  material
                                                        DEF _*GLOBAL* Material {
                                                          ambientIntensity        0.666667
                                                          diffuseColor    1 1 0
                                                          specularColor   1 1 0
                                                          transparency    0
                                                        }
                                                }
                                          geometry
                                                IndexedFaceSet {
                                                  coord
                                                        Coordinate {
                                                          point  [ AutoCAD output omitted here]
```

```
                        }
                normal
                        Normal {
                          vector[ AutoCAD output omitted here]
                        }
                texCoord
                        TextureCoordinate {
                          point  [ AutoCAD output omitted here ]
                        }
                solid  FALSE
                creaseAngle    3.14159
                coordIndex     [ AutoCAD output omitted here ]
              }
          }
        ]
      }
  ]
},
DEF INTWALLS Transform {
  children [
      Transform {
        children [
          Shape {
            appearance
              Appearance {
                material
                  DEF _*GLOBAL*01 Material {
                    ambientIntensity        0.333333
                    diffuseColor   0 0 1
                    specularColor  0 0 1
                    transparency   0
                  }
              }
            geometry
              IndexedFaceSet {
                coord
                  Coordinate {
                    point  [ AutoCAD output omitted here ]
                  }
                normal
                  Normal {
                    vector[ AutoCAD output omitted here ]
                  }
                texCoord
                  TextureCoordinate {
                    point  [ AutoCAD output omitted here ]
                  }
                solid  FALSE
                creaseAngle    3.14159
                coordIndex     [ AutoCAD output omitted here ]
              }
          }
        ]
      }
  ]
```

```
},
DEF FLOOR Transform {
  children [
        Transform {
          children [
                Shape {
                  appearance
                        Appearance {
                          material
                                DEF _*GLOBAL*02 Material {
                                  ambientIntensity        0.576471
                                  diffuseColor   0.576471 0.576471 0.576471
                                  specularColor  0.576471 0.576471 0.576471
                                  transparency   0
                                }
                        }
                  geometry
                        IndexedFaceSet {
                          coord
                                Coordinate {
                                  point  [ AutoCAD output omitted here ]
                                }
                          normal
                                Normal {
                                  vector[ AutoCAD output omitted here ]
                                }
                          texCoord
                                TextureCoordinate {
                                  point  [ AutoCAD output omitted here ]
                                }
                          solid  FALSE
                          creaseAngle    3.14159
                          coordIndex     [ 0, 1, 2, -1, 1, 3, 2, -1,
                                              4, 5, 6, -1, 5, 7, 6, -1,
                                              8, 9, 10, -1, 8, 11, 9, -1,
                                              12, 13, 14, -1, 13, 15, 14, -1,
                                              16, 17, 18, -1, 17, 19, 18, -1,
                                              20, 21, 22, -1, 21, 23, 22, -1 ]
                        }
                }
          ]
        }
  ]
},
DEF ROOF Transform {
  children [
        Transform {
          children [
                Shape {
                  appearance
                        Appearance {
                          material
                                DEF _*GLOBAL*03 Material {
                                  ambientIntensity        0
                                  diffuseColor   0 0.5 0.5
```

```
                                        specularColor  0 0 0
                                        transparency    0
                                    }
                                }
                            geometry
                                IndexedFaceSet {
                                    coord
                                        Coordinate {
                                            point  [ AutoCAD output omitted here ]
                                        }
                                    normal
                                        Normal {
                                            vector [ AutoCAD output omitted here ]
                                        }
                                    texCoord
                                        TextureCoordinate {
                                            point  [ AutoCAD output omitted here ]
                                        }
                                    solid  FALSE
                                    creaseAngle    3.14159
                                    coordIndex     [ AutoCAD output omitted here ]
                                }
                        }
                    ]
                }
    ]
},
DEF WALLS Transform {
    children [
            Transform {
                children [
                        Shape {
                            appearance
                                Appearance {
                                    material
                                        DEF _*GLOBAL*04 Material {
                                            ambientIntensity          0.333333
                                            diffuseColor    1 0 0
                                            specularColor  1 0 0
                                            transparency    0
                                        }
                                }
                            geometry
                                IndexedFaceSet {
                                    coord
                                        Coordinate {
                                            point  [ AutoCAD output omitted here ]
                                        }
                                    normal
                                        Normal {
                                            vector [ AutoCAD output omitted here ]
                                        }
                                    texCoord
                                        TextureCoordinate {
                                            point  [ AutoCAD output omitted here ]
```

```
                                    }
                                solid  FALSE
                                creaseAngle    3.14159
                                coordIndex     [ AutoCAD output omitted here ]
                            }
                        }
                    ]
                }
            ]
        },
        DEF FRONTDOOR Transform {
            children [
                Transform {
                    children [
                        Shape {
                            appearance
                                Appearance {
                                    material
                                        DEF _*GLOBAL*05 Material {
                                            ambientIntensity        0.333333
                                            diffuseColor    0 1 0
                                            specularColor  0 1 0
                                            transparency    0
                                        }
                                }
                            geometry
                                IndexedFaceSet {
                                    coord
                                        Coordinate {
                                            point  [ AutoCAD output omitted here ]
                                        }
                                    normal
                                        Normal {
                                            vector [ AutoCAD output omitted here ]
                                        }
                                    texCoord
                                        TextureCoordinate {
                                            point  [ AutoCAD output omitted here ]
                                        }
                                    solid  FALSE
                                    creaseAngle    3.14159
                                    coordIndex     [ AutoCAD output omitted here ]
                                }
                        }
                    ]
                }
            ]
        }
    ]
  }
 ]
}
]
}
```